Introduction to Artificial Intelligence

State Space Search

Gregory Adam

Contents

- Intro
- Theory
  - State Space Search
    - Blind State Space Search (3 algorithms)
    - Heuristic State Space Search (2 algorithms)
- Example

Most of the material used (except the examples) comes from
“The Handbook of Artificial Intelligence – Volume I”
(Avron Barr & Edward A Feigenbaum)

What

- Part of Computer Science concerned with designing intelligent computer systems
- Systems exibiting the characteristics we associate with intelligence in human behaviour

Areas

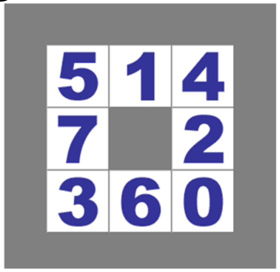
- *The areas are not distinct – most are interrelated*
- Problem Solving
  - Puzzles
  - Play games, eg chess
  - Symbolic integration of mathematical formulas
  - Some programs can improve their performance with experience
- Logical reasoning
  - Prove assertions (theorems) by manipulating a database of facts
- Language
  - Understanding of natural language
  - Translation
  - Eg Spelling checker
- Programming
  - Write computer programs based on a description

Areas – cont'd

- Learning
  - Learning from examples
- Expertise (aka Expert Systems)
  - User interacts with an Expert System via a dialogue
  - Expert feeds knowledge
- Robotics and vision
  - Manipulate robot devices (mostly in industrial applications to perform repetitive tasks)
  - Recognize objects and shadows in visual scenes
- Systems and languages
  - Time-sharing, list processing, and interactive debugging were developed in the AI research

Search

- Components of search systems
  - Database : describes the current task domain and the goal
  - Set of operators: transform one state to another
    - Eg in the 8 puzzle: UP, DOWN, LEFT, RIGHT
  - Control strategy : decides what to do next



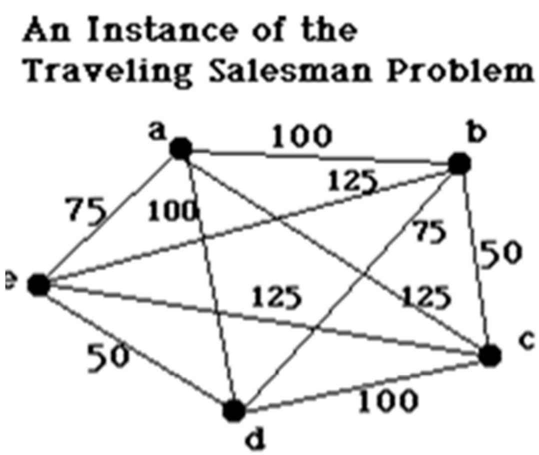
- Definition
  - Find a finite sequence of operators transforming the initial state to a goal state
- Reasoning
  - Forward : Transform original state to a goal state
  - Backward: Transform a goal state to the original state

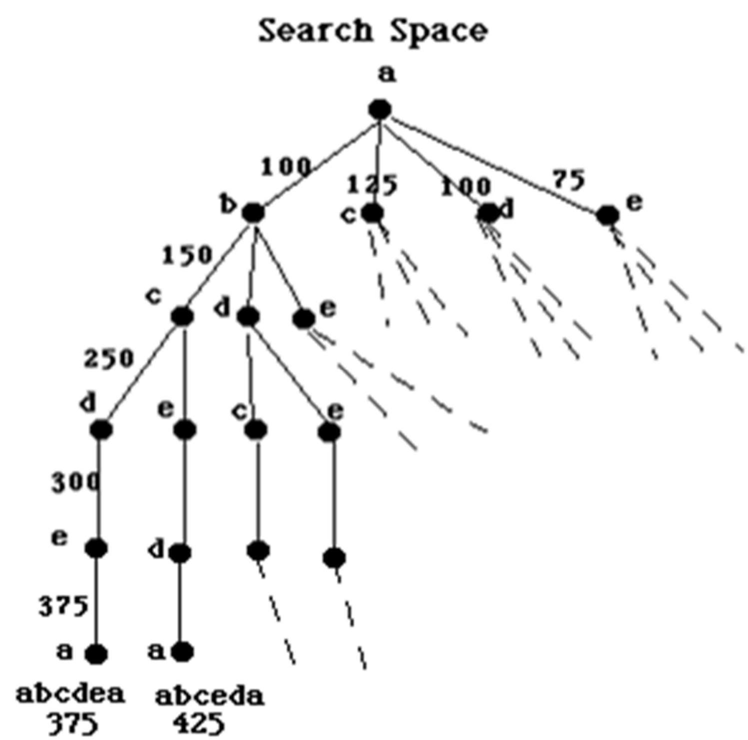
Search

- State Space and Problem Reduction
  - State space
    - An operator produces exactly one new state
  - Problem reduction
    - An operator produces a set of subproblems, each of which have to be solved
    - Eg
      - Tower of Hanoi
      - Integrate (f(x) + g(x)) dx
        - » Integrate f(x) dx
        - » Integrate g(x) dx
        - » Add the results

Search - problem representation

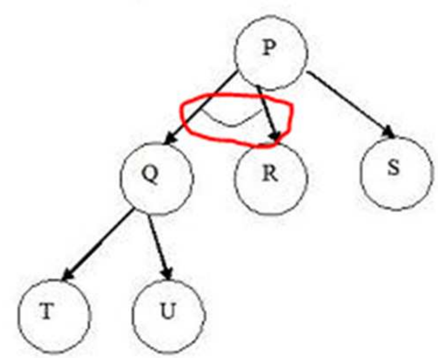
- State Space: State space graph





Search - problem representation

- AND/OR Graph
  - Horizontally connected edges (here marked in red) represent AND nodes
  - For AND nodes, each of the nodes have to be solved
  - Eg
    - problem reduction
    - Games (eg chess)

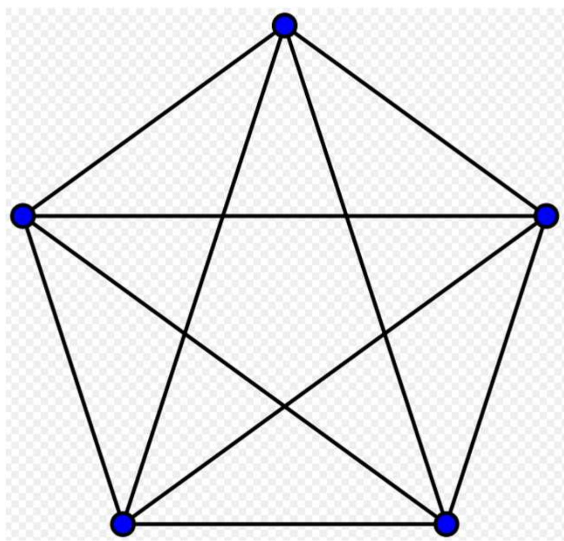


Search – Blind Search

- The Blind search algorithms following
  - Breadth First Search
  - Depth First Search
  - Uniform Cost Search
  – Assume the State Space graph is a Directed Tree
- The Heuristic search algorithms following
  - Ordered Search
  - A* An optimal search for an optimal solution
  – Assume the State Space graph is a General Graph

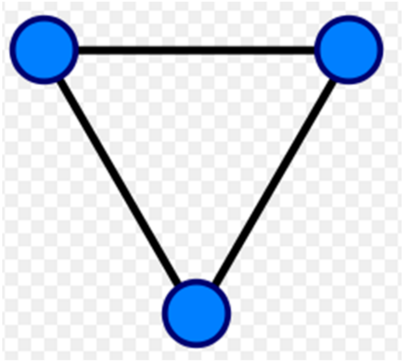
Search

- General graph consists of
  - Nodes or points
  - Arcs or edges connecting two nodes



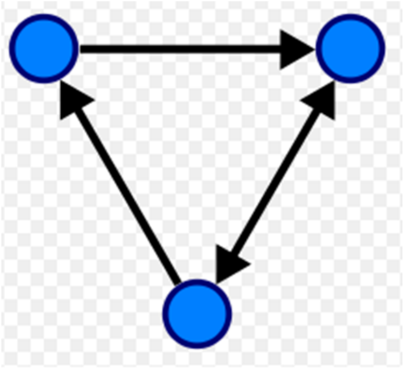
Search

- Arcs can be
  - Undirected



    or

  - Directed



Search

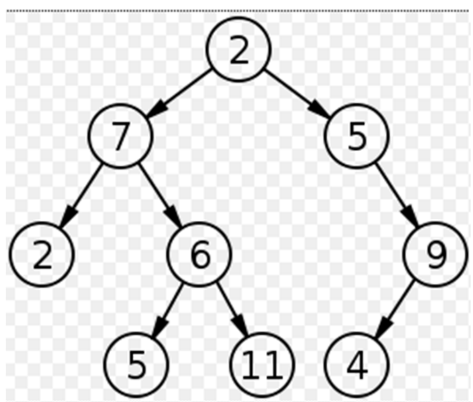
- Undirected graph
  - Contains only undirected arcs
- Directed graph or digraph
  - Contains only directed arcs
- Mixed graph
  - Contains both directed and undirected arcs

Search

- In a directed graph (containing only directed arcs)
- The indegree of a node
  - Is the number of arcs terminating in that node
- The outdegree of a node
  - Is the number of arcs starting in that node

Search

- A Directed tree
  - Is an acyclic digraph
    - Which has one node called the root
      - The root node has indegree zero, and
    - All other nodes have indegree one

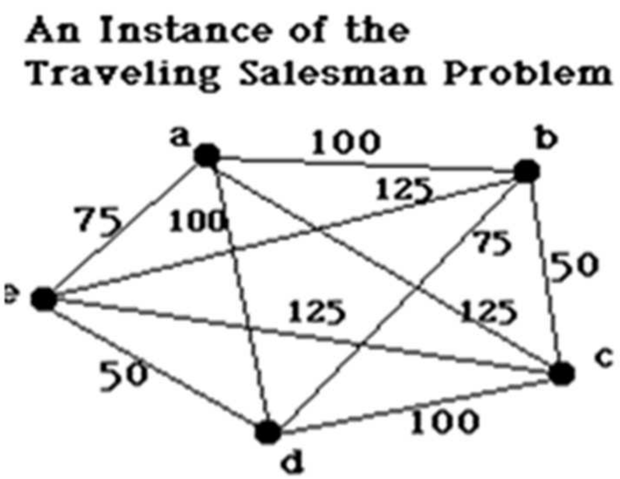


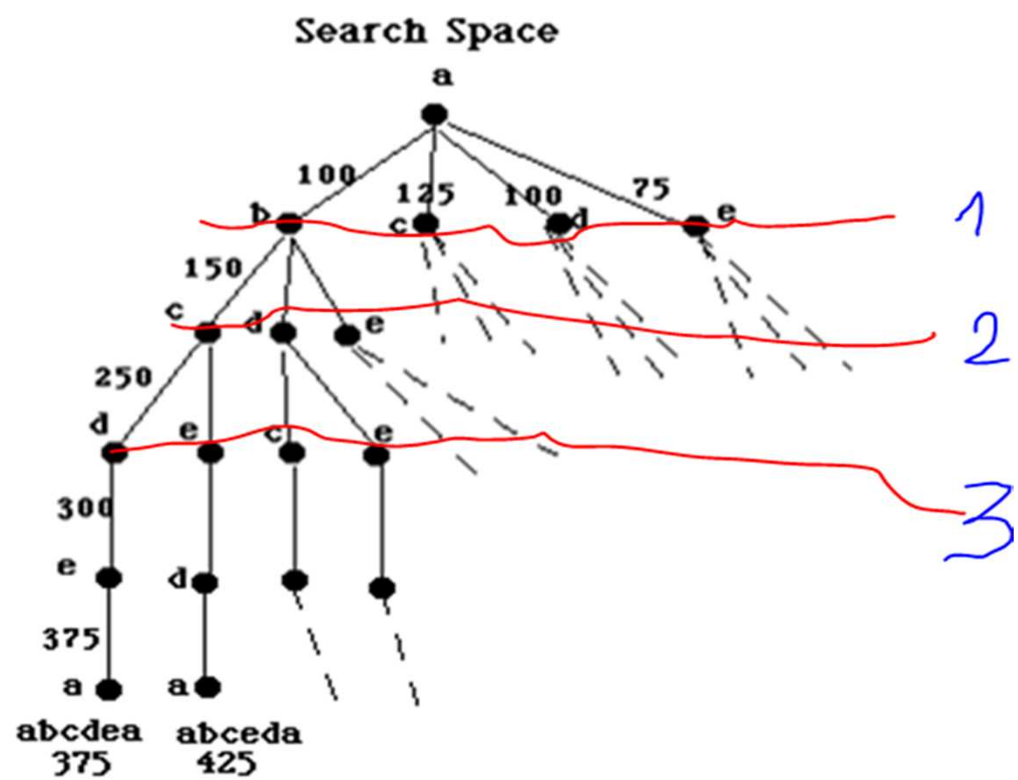
Search – Blind State Space Search
Breadth-First Search

- Expands nodes in their proximity from the root (or start) node
- Expands all nodes of depth n before expanding nodes of depth n+1
- Guaranteed to find the shortest possible solution

Search – Blind State Space Search

Breadth-First Search



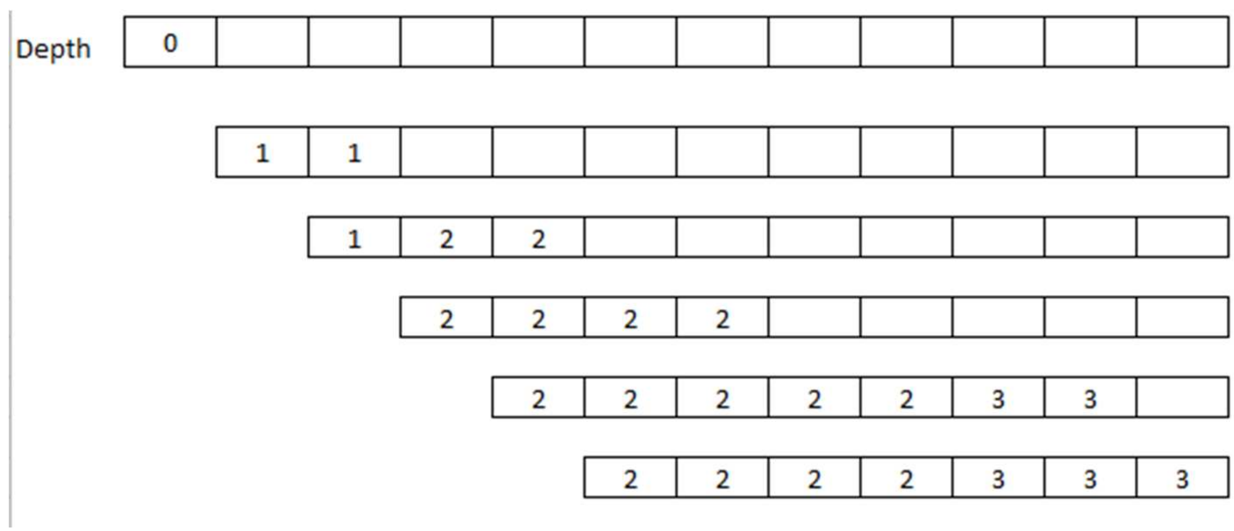


Search – Blind State Space Search
Breadth-First Search - Algorithm

1. Put the start node on a list, called OPEN, of unexpanded nodes
2. If OPEN is empty, no solution exists
3. Remove the first node, n, from OPEN and put it in a list, called CLOSED, of expanded nodes
4. Expand node n. If it has no successors, go to (2)
5. Place all successors of n at the end of the OPEN list
6. If any of the successors of n is a goal node, a solution has been found. Otherwise go to (2)

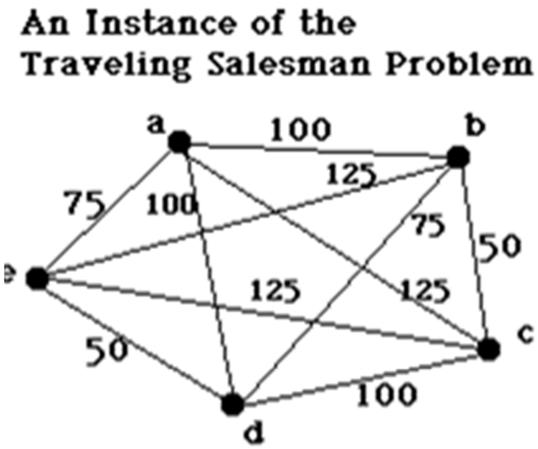
Search – Blind State Space Search
Breadth-First Search - Algorithm

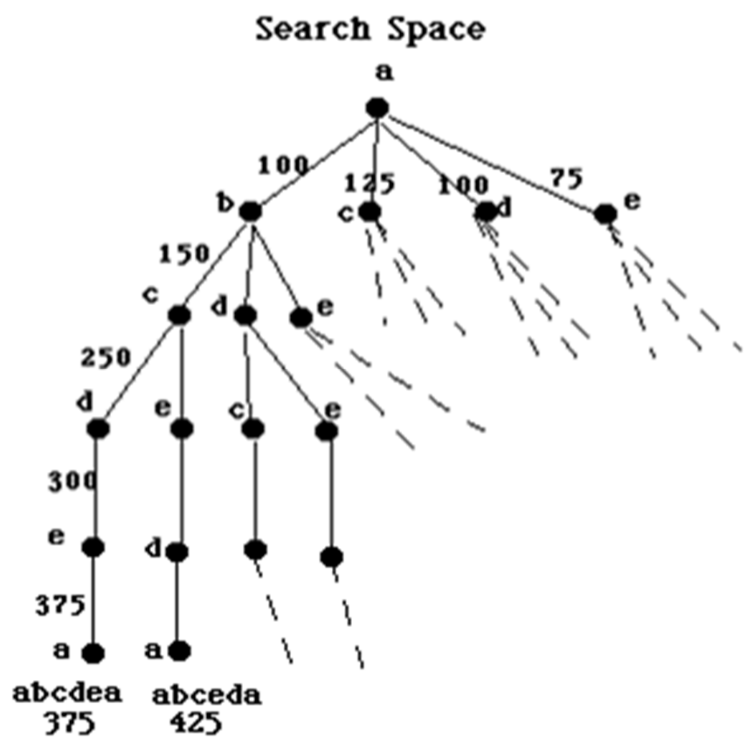
Newly expanded nodes are added to the end of the list



Search – Blind State Space Search
Depth-First Search

- Expands most recent (deepest) nodes first
  - Here abcdea first



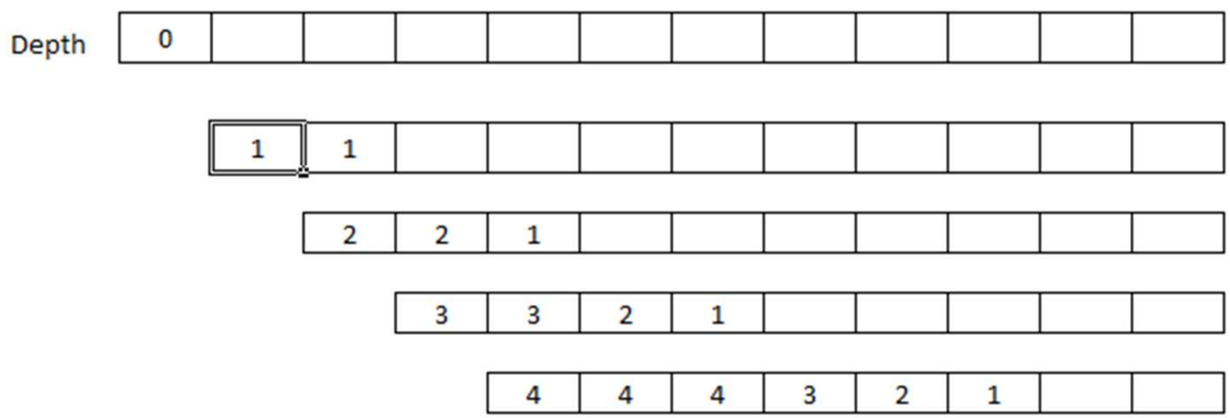


Search – Blind State Space Search
Depth-First Search - Algorithm

1. Put the start node on a list, called OPEN, of unexpanded nodes
2. If OPEN is empty, no solution exists
3. Remove the first node, n, from OPEN and put it in a list, called CLOSED, of expanded nodes
4. Expand node n. If it has no successors, go to (2)
5. *If the depth of node n is greater than the maximum depth, go to (2)*
6. Place all successors of n at the beginning of OPEN list
7. If any of the successors of n is a goal node, a solution has been found. Otherwise go to (2)

Search – Blind State Space Search
Depth-First Search

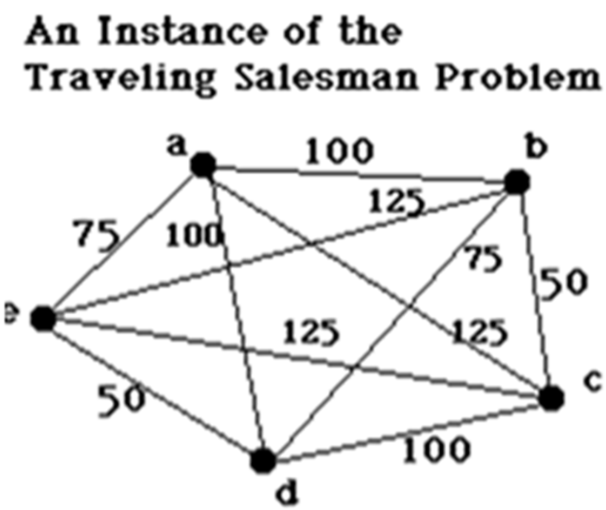
- Newly expanded nodes are added at the beginning of the list

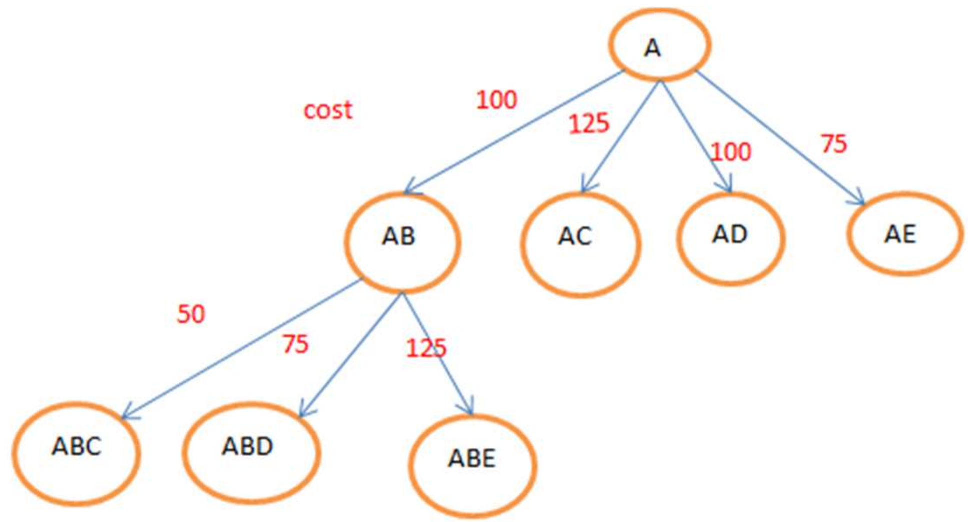


Search – Blind State Space Search
Uniform Cost Search

- The Breadth-First search can be generalized slightly to solve to problem of finding the cheapest path from a start node to a goal state
- A non-negative cost is associated with each arc joining two nodes
- The cost of a solution is then the sum of all the costs along the path

Search – Blind State Space Search
Uniform Cost Search





Search – Blind State Space Search
Uniform Cost Search - Algorithm

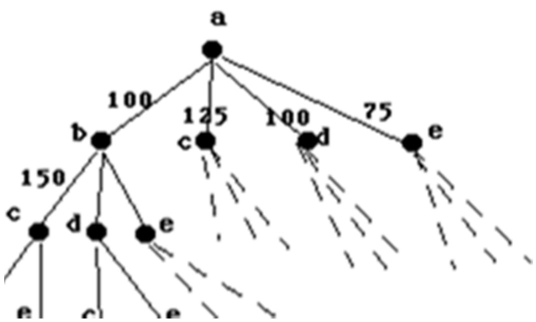
1. Put the start node on a list, called OPEN, of unexpanded nodes
2. If OPEN is empty, no solution exists
3. Select from OPEN a node *i* such that TotalCost(*i*) is minimum. If several nodes qualify choose *i* to be a goal node if there is one, otherwise choose among them arbitrarily.
4. Remove node *i* from OPEN and place it on a list CLOSED of expanded nodes
5. If node *i* is a goal node, a solution has been found
6. Expand node *i*, if it has no successors go to (2)
7. For each successor node *j* of *I*
   - Compute TotalCost(*j*) = TotalCost(*i*) + Cost(*j*)
   - Add node *j* to the OPEN list
8. Go to (2)

Search – Blind State Space Search
Uniform Cost Search

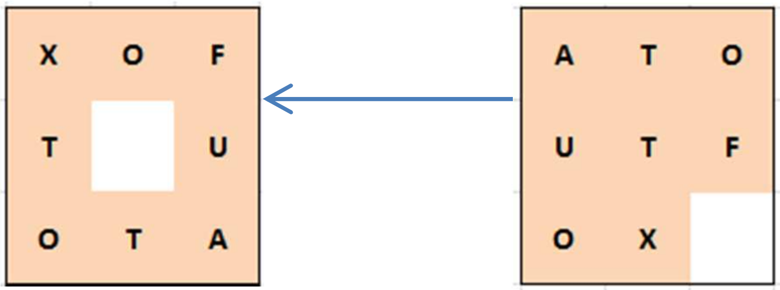
- If we associate the cost of node *i* to node *j* with
  - -1
    - the Uniform Cost Search becomes a Depth-First Search
    - Since TotalCost of the node = - Depth of the node
  - 1
    - the Uniform Cost Search becomes a Breadth-First Search
    - Since TotalCost of the node = Depth of the node

Search – Blind State Space Search
Bidirectional Search

- The algorithms so far use forward reasoning, ie moving from the start node towards a goal node



- In some cases we could use backward reasoning, ie moving from the goal state to the start state



Search – Blind State Space Search
Bidirectional Search

- Forward and backward reasoning can be combined into a technique called bidirectional search
- The idea is to replace a single search graph – which is likely to grow exponentially – by two smaller graphs
  - One starting from the initial state and searching forward
  - One starting from the goal state and searching backward

  The search terminates when the two graphs intersect

Search – Blind State Space Search

Bidirectional Search

- A bidirectional version of the Uniform Cost Search, guaranteed to find the shortest solution path, is due to Pohl (1969, 1971)
- Empirical data for randomly generated graphs expanded only ¼ as many nodes as unidirectional search

Search – Limiting the search

- The amount of time and space is critical to find a solution
  - Heuristics
  - Relaxing the requirement
    - Any (fast) solution, but not necessarily the best

Search - Heuristics

- In blind search the number of nodes can be extremely large
  - The order of expanding the nodes is arbitrary
  - Blind search does not use any properties of the problem being solved
  - Result is the combinatorial explosion
- Information about a particular problem can help to reduce the search
  - The question then is: how to search the given space efficiently

Search - Heuristics

- Heuristic information
  - Additional information beyond that which is built into the state and operator definitions
- Heuristic search
  - A search method using that heuristic information
  - Whether or not the method is foolproof
- Most of the programs were written for a single domain – heuristics were closely intertwined in the program and not accessible for study and adaptation to new problems

Search - Heuristics

- Heuristic search
  - Strategy to limit (drastically) the search for solutions in large problem spaces
- Ways of using heuristic information
  - Which node(s) to expand first instead of expanding is a strictly depth-first or breadth-first manner
  - When expanding a node, decide which successors to generate instead of blindly generate all successors at one time
  - Which nodes not to expand at all (pruning)

Search – Heuristics
Ordered or Best-Fit Search

- Addresses only the first point
  - Which node to expand first
  - Expands fully
  - The idea is to expand the node that seems most promising
  - The promise of a node can be defined in several ways
    - Estimate its distance to the goal node
    - Estimate the length of the entire path
  - In all cases the measure of promise of a node is estimated by calling an evaluation function

Search – Heuristics Ordered or Best-Fit Search

- The basic algorithm is given by Nilsson (1971)
- The evaluation function f^* is defined so that the more promising a node is, the smaller is the value of f^*
  - Estimates its distance to the goal node
- The node selected for expansion is the one at which f^* is minimum
- The search space is assumed to be a general graph

Ordered or Best-Fit Search - Algorithm

1. Put the start node s on a list called OPEN of unexpanded nodes. Calculate $f^*(s)$ and associate its value with node s
2. If OPEN is empty, exit with failure; no solution exists
3. Select from OPEN a node i such that $f^*(i)$ is minimum. If several nodes qualify choose i to be a goal node if there is one, otherwise choose among them arbitrarily.
4. Remove node i from OPEN and place it on a list CLOSED of expanded nodes
5. If i is a goal node, exit with success; a solution has been found

(continued on next slide)

Ordered or Best-Fit Search - Algorithm

6. Expand node *i*
7. For each successor node *j* of *i*
   a. Calculate *f**(*j*)
   b. If *j* is neither in list OPEN or list CLOSED, add it to OPEN. Attach a pointer back from *j* to its predecessor *i*
   c. If *j* was already on OPEN or CLOSED, compare the *f** value just calculated with the previously calculated value
   d. If the new value is lower
      i. Substitute it for the old value
      ii. Point *j* back to *i* instead of to its previously found predecessor
      iii. If *j* was on the CLOSED list, move it back to OPEN
8. Go to (2)

A* Optimal search for an optimal solution

- Ordered search looked only at the promise of the node, not necessarily at the minimum cost or path
- We can change f* slightly to find a minimum cost solution
- $f^*(n) = g^*(n) + h^*(n)$
  - $g^*(n)$ estimates the minimum cost of the start node to n
  - $h^*(n)$ estimates the minimum cost of n to the goal
- h*
  - is the carrier of heuristic information
  - Should never overestimate the cost
    - $h^*(n) \le h(n)$

Example – Breadth-First Search

- Given a number of amounts, try to find a combination of amounts that matches another amount
- Example
  - We have 10, 20, 30, 40, ….., 100
  - Find all combinations that produce eg 120
- We use Breadth-First search, ie find first those with a minimum of depth – more likely

Total number of combinations

- If we have n amounts
- $\sum_{p=0}^{n} Comb(n, p)$
  - Comb(n, p) = n! / ((n-p)! * p!)
- Example with 50 amounts
  - $\sum_{p=0}^{50} Comb(50, p)$ = 1,125,899,906,842,623

Heuristics

- Heuristic information
  - Sort amounts in ascending sequence
  - For each index
    - Calculate minimum sum ahead
    - Calculate maximum sum ahead
- Heuristic search
  - Prune as soon as possible

Index	1	2	3	4	5	6	7	8	9
Amounts	-30	-20	-10	40	50	60	70	80	90
Minimum sum ahead	-60	-30	-10	40	50	60	70	80	90
Maximum sum ahead	390	390	390	390	350	300	240	170	90

LastNegativeIndex	3

Heuristics

- Construction of Min/Max sum ahead

Index	1	2	3	4	5	6	7	8	9
Amounts	-30	-20	-10	40	50	60	70	80	90
Minimum sum ahead	-60	-30	-10	40	50	60	70	80	90
Maximum sum ahead	390	390	390	390	350	300	240	170	90

LastNegativeIndex 3

Minimum sum ahead
```
- copy Amounts
for i = LastNegativeIndex - 1 to 1
        [ i ] = [ i ] + [ i + 1 ]
endfor
```

Maximum sum ahead
```
- copy Amounts
for i = maxIndex - 1 to LastNegativeIndex + 1
        [ i ] = [ i ] + [ i + 1 ]
endfor

for i = LastNegativeIndex  to 1
        [ i ] = [ i + 1 ]
endfor
```

Heuristics

Index	1	2	3	4	5	6	7	8	9
Amounts	-30	-20	-10	40	50	60	70	80	90
Minimum sum ahead	-60	-30	-10	40	50	60	70	80	90
Maximum sum ahead	390	390	390	390	350	300	240	170	90

LastNegativeIndex	3

(1) Search for 500

500 is not between the min and max of the first index

We don't even have to start searching - pruned all 511 combinations

(2) Search for 70 with 4 combinations	look for	index	Value	MinAhead	MaxAhead	between
We have index 4, ie 40	70	4	40	40	390	yes
Continue searching from index 5 onwards	30	5	50	50	350	no

still need 70-40 or 30

30 is not between 50 and 350

prune : comb(9-5+1, 4-2) =10

Heuristics

- Formula to calculate how many nodes are going to be pruned

Index	1	2	3	4	5	6	7	8	9
Amounts	-30	-20	-10	40	50	60	70	80	90
Minimum sum ahead	-60	-30	-10	40	50	60	70	80	90
Maximum sum ahead	390	390	390	390	350	300	240	170	90

General pruned formula

n = number of amounts in array

maxDepth = maximum depth investigating

currentDepth = current depth

investigatingIndex : index investigating for further expansion

pruned = comb(n - investigatingIndex + 1, maxDepth - currentDepth)

Example (2) of previous page

n = 9

maxDepth = 4

currentDepth = 1

investigatingIndex = 5

comb(9 - 5 + 1, 4 - 1)

Heuristics

- Sign Reversal : try to prune as early as possible
- Decrease the gaps between minimum sum ahead and maximum sum ahead as soon as possible
- Make sure the number with the highest absolute value is at the front

- Reverse the sign of all amounts + the amount to search if necessary

Heuristics

Index	1	2	3	4	5	6
Amounts	-30	-20	-10	1010	1020	1030
Minimum sum ahead	-60	-30	-10	1010	1020	1030
Maximum sum ahead	3060	3060	3060	3060	2050	1030
Gap Minimum/Maximum	3120	3090	3070	2050	1030	0
Gap Delta		30	20	1020	1020	1030

Biggest abs() in front

Index	1	2	3	4	5	6
Amounts	-1030	-1020	-1010	10	20	30
Minimum sum ahead	-3060	-2030	-1010	10	20	30
Maximum sum ahead	60	60	60	60	50	30
Gap Minimum/Maximum	3120	2090	1070	50	30	0
Gap Delta		1030	1020	1020	20	30

Heuristics

Index	1	2	3	4	5	6	7	8	9	10
Amounts	1	2	3	4	5	6	7	8	9	10
Minimum sum ahead	1	2	3	4	5	6	7	8	9	10
Maximum sum ahead	55	54	52	49	45	40	34	27	19	10
Gap Minimum/Maximum	54	52	49	45	40	34	27	19	10	0
search	5	5	5	5	5	5	5	5	5	5
take	1	2	3	4	5	6	7	8	9	10
new find	4	3	2	1	0	-1	-2	-3	-4	-5

Index	1	2	3	4	5	6	7	8	9	10
Amounts	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
Minimum sum ahead	-55	-45	-36	-28	-21	-15	-10	-6	-3	-1
Maximum sum ahead	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
Gap Minimum/Maximum	54	44	35	27	20	14	9	5	2	0
search	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5
take	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
new find	5	4	3	2	1	0	-1	-2	-3	-4